A Microservice-Based Framework for Multi-Domain SDN Orchestration through Controller Decomposition

Yasin Saedi†, Gianluca Davoli†, Domenico Scotece⋆, Carla Raffaelli†, Walter Cerroni†, Luca Foschini⋆
†Department of Electrical, Electronic, and Information Engineering "G. Marconi", University of Bologna, Italy
⋆Department of Computer Science and Engineering, University of Bologna, Italy
Email: {name.surname}@unibo.it

***Abstract*—Traditional SDN controllers are usually deployed as monolithic systems or tightly coupled service chains, limiting their adaptability in distributed or federated network domains. This paper presents eMSN, a microservice-based SDN framework that enables controller decomposition and decentralized multi-domain orchestration, providing a foundation for scalable, and domain-aware SDN experimentation. The framework introduces lightweight, containerized microservices that interact via REST APIs and coordinate through a shared ETCD cluster. The main one, called FlowBlocker, collects topology and host information from the emitter, builds a policy-aware decision table, and shares domain-scoped state in ETCD. The latter stores only host/topology data and never flow rules; enforcement decisions remain within FlowBlocker, which installs rules locally via Ryu Core or coordinates with peer FlowBlockers across domains. The architecture supports centralized, partially decentralized, and fully decentralized deployment models. The proof-of-concept implementation uses Docker and Mininet for reproducibility. Functional evaluation demonstrates sub-millisecond Packet-In responsiveness, tens-of-milliseconds policy enforcement latency, and correct blocking of unauthorized traffic.**

***Index Terms*—Software-Defined Networking; Microservices; Multi-Domain Orchestration.**

I. Introduction

Software-Defined Networking (SDN) has redefined network control by decoupling the control plane from the data plane and enabling programmability through logically centralized controllers [1]. However, traditional controller architectures are often *monolithic and tightly integrated*, which creates challenges in scalability, maintainability, and extensibility, especially in *multi-domain* and *edge-cloud* environments [2], [3]. As networks become increasingly distributed and heterogeneous, with optical technologies supporting unprecedented performance levels for the interconnection of multiple domains [4], the need for *controller decomposition* (i.e., breaking monolithic controllers into modular, independently deployable microservices) has emerged as a compelling design shift [5], [6].

Existing systems, such as ONOS and OpenDaylight, offer modular designs but still rely on centralized orchestration or tightly coupled components, which limits adaptability in multi-domain scenarios [6], [7]. Hierarchical control architectures [8] provide domain abstraction but cannot fully support autonomous or decentralized policy enforcement across domains.

This paper introduces **eMSN** (evolved Microservice-based SDN Network), a framework that enables decomposed SDN control and decentralized orchestration across multiple domains. eMSN extends our prior MSN framework [9] by incorporating a domain-aware coordination layer based on *ETCD*, a strongly consistent distributed key-value store, and a dedicated enforcement service called *FlowBlocker*. Unlike prior systems, eMSN does not rely on centralized policy resolution or ETCD-stored flow rules. Instead, each FlowBlocker collects topology and host information via the Emitter, builds an internal policy-aware decision table, and shares state in ETCD under domain-specific namespaces. When enforcement is required, FlowBlocker installs OpenFlow rules through its local Ryu Core or coordinates with peer FlowBlockers, preserving decentralized decision-making.

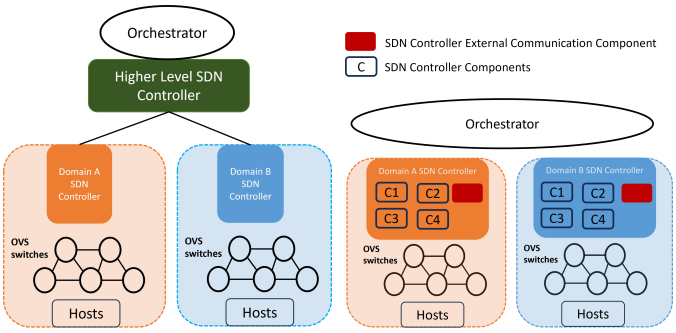
Each SDN control function (topology discovery, forwarding, and policy enforcement) is implemented as a standalone containerized microservice. The Ryu Core itself is decomposed into a REST client and an Emitter, the latter handling OpenFlow event collection and distribution. This modular design supports fault isolation, domain-scoped deployment, and reproducibility. In contrast to other SDN microservice platforms [5], [10], eMSN prioritizes multi-domain orchestration and uses ETCD strictly for topology and host synchronization, not for flow control.

The contributions of this paper are as follows:

- a decomposed SDN framework (eMSN) based on microservices, enabling modular and domain-scoped control;
- a decentralized policy enforcement mechanism (FlowBlocker) that operates autonomously while coordinating across domains;
- a reproducible testbed implementation using Docker, Mininet, and ETCD for orchestrating microservice deployments;
- functional validation with metrics that quantify controller responsiveness and policy enforcement behavior.

This paper presents the conceptual design and functional implementation of eMSN as a research platform for scalable,

978-3-903176-75-1 ©2025 IFIP



(a) Traditional (b) Decomposed

Fig. 1: Multi-domain SDN orchestration.

and domain-aware SDN orchestration. While large-scale performance evaluation is left for future work, the architecture provides a foundation for further research in trust-aware coordination, distributed policy enforcement, and intelligent orchestration.

II. Background and Motivation

A. Controller Decomposition in SDN

SDN separates the control and data planes, introducing programmability into network infrastructures. Traditional SDN controllers, such as ONOS and OpenDaylight, are generally designed as monolithic systems in which topology discovery, policy enforcement, and flow management are tightly coupled within a single software stack [2], [3]. While this simplifies integration, it introduces limitations in scalability, maintainability, and resilience.

Monolithic controllers typically operate as centralized entities, where a fault or update in one component can affect the entire system. In geographically distributed environments, relying on a single centralized control stack becomes impractical, especially when domains require different update cycles or responsiveness levels. This limitation is illustrated in Fig. 1a, showing a hierarchical SDN deployment where both top-level and domain-level controllers remain monolithic.

To address such rigidity, recent efforts have explored controller decomposition into modular subsystems. Instead of treating the controller as a black box, functions such as topology management, reactive forwarding, and policy enforcement can be disaggregated and deployed independently [5], [6]. This improves fault isolation, testing, and scalability. Fig. 1b shows decomposed domain controllers. However, even in this model, coordination is still governed by a central orchestrator, which preserves hierarchical coupling.

B. Challenges in Multi-Domain SDN Orchestration

While decomposition improves intra-domain modularity, the broader challenge is coordinating across multiple domains. Real deployments often involve administrative boundaries, cloud–edge integration, or federated providers, each with its own topology, policy, and trust assumptions [7]. The prevailing solution is hierarchical orchestration, where a central entity interfaces with domain controllers via standardized northbound APIs [8].

This provides a unified control view but reduces domain autonomy, introduces bottlenecks and single points of failure, and assumes homogeneous interfaces and semantics. Even with decomposed controllers (Fig. 1b), the reliance on a central orchestrator leaves cross-domain coordination globally coupled.

C. Toward Decentralized Coordination with Shared State

Modern distributed systems overcome similar issues using strongly consistent key-value stores (e.g., ETCD, Consul) for coordination without centralized controllers. ETCD offers hierarchical namespaces, publish–subscribe updates via `watch`, and state replication with Raft consensus [9], [10].

Applying this paradigm to SDN enables decentralized control: domain-specific microservices can subscribe to host/topology updates, react to policy changes, and enforce rules locally—without relying on global rule computation. This paper builds on this concept by introducing eMSN, where state dissemination is decoupled from enforcement. FlowBlocker consumes topology and host information from the Emitter, publishes summaries into ETCD under domain-specific namespaces, and enforces policies by installing rules via the local Ryu Core or coordinating with peer FlowBlockers. Crucially, flow rules are never stored in ETCD. This design provides decentralized enforcement, domain autonomy, and improved fault isolation, motivating the architecture described in the next section.

III. Related Work

The limitations of monolithic SDN controllers have motivated interest in decomposing the control plane into modular components. One prominent effort is μ-ONOS, which re-architects ONOS into a microservice-based controller [6]. μ-ONOS introduces REST/gRPC interfaces and independent scaling, but orchestration remains centralized and tied to the ONOS runtime.

Our prior MSN framework [9] decomposed Ryu into microservices with REST APIs, separating topology discovery and forwarding logic. MSN demonstrated deployment flexibility but lacked a coordination layer: services relied only on REST, and cross-domain scenarios required manual synchronization.

μ-TSN-CP applies a microservice model to Time-Sensitive Networking [10], decomposing scheduling, topology, and monitoring functions to meet deterministic latency goals. However, it is domain-specific and not designed for multi-domain or general-purpose coordination.

Hierarchical orchestration frameworks such as Kandoo [8] separate local and root controllers for scalability, but global decisions remain centralized. MDSO [7] introduces service orchestration across heterogeneous controllers, again depending on global policy engines and standardized NBIs. Both approaches limit domain autonomy.

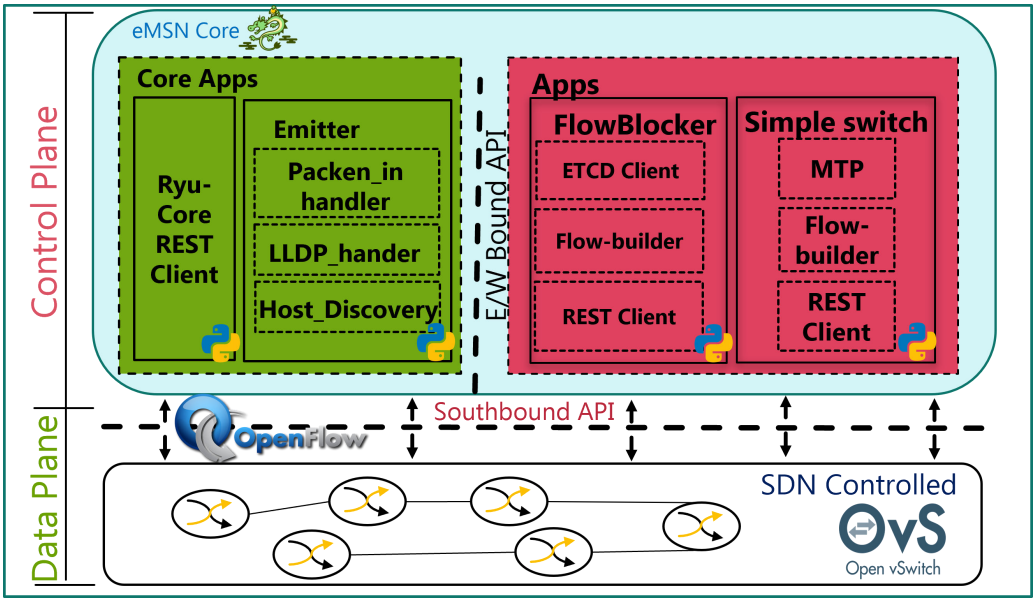


Fig. 2: Overview of the eMSN architecture and its microservice interactions.

In parallel, cloud-native platforms such as Kubernetes demonstrate the utility of consistent key-value stores (ETCD, Consul) for distributed coordination. While widely adopted for container orchestration, these systems are rarely applied to SDN controllers. Some work uses ETCD for switch configuration, but few use it to disseminate runtime host/topology state for decentralized enforcement. Recent efforts in intent-based multi-domain orchestration [11], [12] reinforce the need for designs that combine flexibility with autonomy.

In summary, prior work has advanced both controller decomposition and hierarchical orchestration, but the combination of domain-scoped microservices with decentralized policy enforcement via a shared state layer remains underexplored. eMSN addresses this gap by enabling multi-domain orchestration with FlowBlocker-based enforcement, while using ETCD strictly for state sharing and never for flow rule storage.

IV. The eMSN Architecture

The eMSN (evolved Microservice-based SDN Controller) framework enables domain-scoped SDN control by decomposing controller logic into independently deployable microservices. Building on MSN [9], eMSN introduces decentralized coordination via ETCD and policy enforcement through FlowBlocker. Fig. 2 provides an overview of the architecture.

Each domain runs three microservices, namely **Ryu Core**, **SimpleSwitch**, and **FlowBlocker**. These services interact via REST APIs and may optionally share state through ETCD. Each is containerized for modular deployment, allowing centralized, partially decentralized, or fully decentralized orchestration.

To start with, Ryu Core consists of an *emitter* and a *REST client*. The emitter listens for OpenFlow `packet_in`, LLDP, and ARP events from OVS and dispatches them via HTTP to services such as SimpleSwitch or FlowBlocker. The REST client exposes endpoints for installing flows. Ryu Core is stateless, acting only as a dispatcher and installer.

Then, SimpleSwitch implements reactive MAC-learning. It receives `packet_in` events from the emitter, applies forwarding logic, and returns flow installation requests to Ryu Core. It operates independently of FlowBlocker, providing default, policy-free forwarding.

Finally, FlowBlocker is the enforcement microservice. It consumes host and topology data from the emitter, builds a decision table, and publishes summaries to ETCD under domain-specific keys (e.g., `/domain/a/hosts/`). Importantly, ETCD stores only state information and never flow rules, which prevents unauthorized manipulation of forwarding entries through the shared store and confines enforcement decisions to trusted FlowBlocker instances. When a policy trigger occurs, FlowBlocker either (i) installs rules via its local Ryu Core or (ii) sends a REST request to a peer FlowBlocker for remote enforcement.

A distributed ETCD cluster provides consistent state dissemination across domains. Each domain writes to its own prefixed namespace (e.g., `/domain/b/switches/`), enabling visibility without central coupling. FlowBlocker decisions remain local as ETCD never stores or evaluates flow rules.

All microservices communicate via REST and run in Docker containers for isolation, reproducibility, and fault tolerance.

A. Multi-Domain Policy Enforcement with FlowBlocker

FlowBlocker enforces policies independently in each domain while coordinating with peers. Each instance processes only state within its domain's ETCD namespace, ensuring autonomy. On a policy trigger (e.g., block traffic from h1 to h8), FlowBlocker updates its table, installs rules via the local

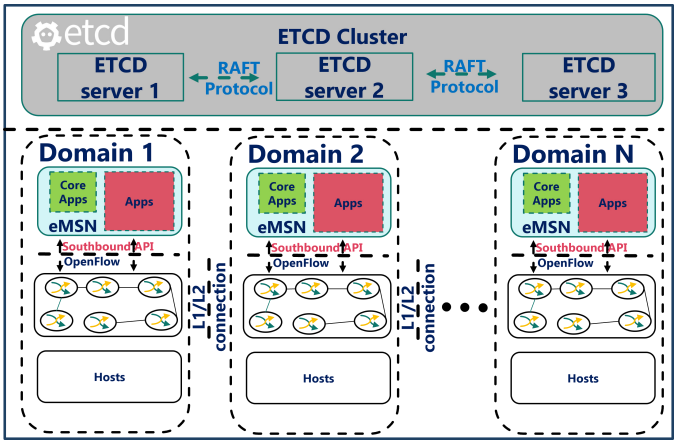


Fig. 3: eMSN in a multi-domain environment, where each domain runs its own stack and shares state via ETCD.

Ryu Core, or sends REST requests to peer FlowBlockers for inter-domain enforcement.

This cooperative model ensures decentralized, domain-aware policy enforcement without relying on ETCD for rule evaluation or a global orchestrator. Fig. 4 illustrates the sequence of policy enforcement, from a client request to flow installation on OVS.

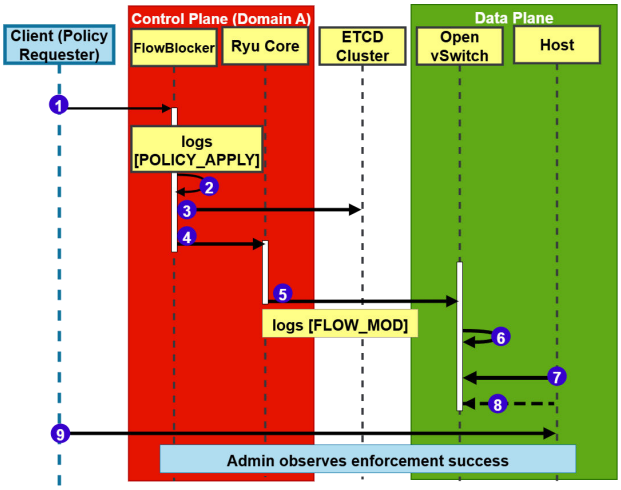


Fig. 4: Sequence of policy enforcement in eMSN. A client issues a block request (1), FlowBlocker applies the decision (2–4), and instructs its local Ryu Core (5). The Ryu Core installs rules on OVS (6), which enforces drops (7–8). The administrator verifies success through logs, dumps, and traffic tests (9).

V. Implementation and Deployment Toolchain

The eMSN framework is implemented as a modular testbed using Docker, Python-based microservices, and Mininet. Each control-plane function runs in its own container, communicating via REST APIs and synchronizing runtime state through a shared ETCD cluster. The system supports centralized, partially decentralized, and fully decentralized deployments, and is designed for reproducibility and experimentation in cloud–edge scenarios.

A. Microservice Components

All components are implemented in Python and expose REST interfaces.

Ryu Core. A minimal controller instance composed of an *emitter* that collects `packet_in`, LLDP, and ARP events and forwards them to external services, and a *REST client* that exposes endpoints for flow installation. It is stateless and performs no forwarding logic, acting solely as dispatcher and installer.

SimpleSwitch. A reactive MAC-learning service that receives events from the emitter, maintains a MAC–port table, and returns flow installation requests to Ryu Core. It is policy-free and operates independently of FlowBlocker.

FlowBlocker. The enforcement service that builds a decision table from emitter-provided host/topology data. It publishes domain-scoped summaries to ETCD but never stores flow rules. Upon a trigger, it installs drop rules via the local Ryu Core or signals a peer FlowBlocker for inter-domain enforcement.

Emitter. Deployed within Ryu Core but architecturally distinct, it extracts host/topology information from OpenFlow events and delivers it to FlowBlocker, providing the live network view across domains.

B. ETCD Coordination Layer

A containerized ETCD cluster serves as the coordination backbone. Each FlowBlocker writes host and switch metadata to a domain-specific namespace, enabling lightweight state sharing without central coupling. Policies may also be exchanged, but enforcement decisions remain inside FlowBlocker. ETCD ensures consistency via the Raft protocol, with isolated namespaces to avoid cross-domain leakage.

C. Containerization and Deployment

All services are packaged as Docker containers and orchestrated via shell scripts. Host networking simplifies REST connectivity with Mininet. Each Mininet domain is mapped to an OVS bridge with its own Ryu Core and FlowBlocker. Traffic is generated using `iperf3` and `ping`, and packet-in events bootstrap discovery.

Deployment is automated through provided scripts that launch the ETCD cluster, instantiate Mininet topologies, and start all microservices. A companion script supports complete environment teardown and cleanup. The framework also provides ready-to-use topology templates, example policy files, and utilities to facilitate reproducibility and extension.

D. Deployment Scenarios

eMSN supports three deployment strategies. In the *centralized* mode, all microservices are co-located on a single host and ETCD is optional. In the *partially decentralized* mode, latency-sensitive components such as the Ryu Core are deployed at the edge, while FlowBlocker remains centralized.

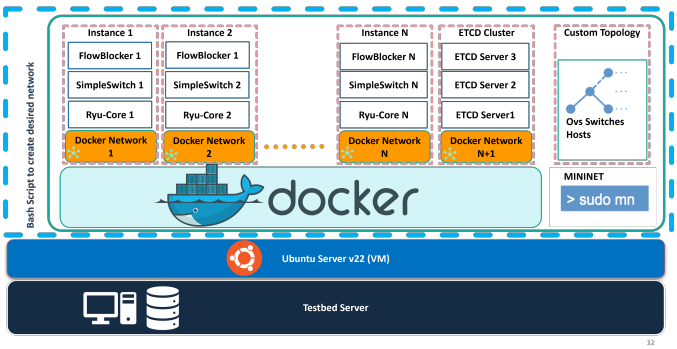


Fig. 5: eMSN testbed deployment with Docker microservices and Mininet data-plane.

TABLE I: Measured metrics for the policy enforcement scenario

Metric	Value	Evidence / Notes
Policy req. to FlowBlocker	15.1 ms	Request epoch vs. `POLICY_APPLY`
Policy req. to first FlowMod	27.0 ms	First "Flow added successfully" log
Packet-In to Flow-Mod	0.278 ms avg.	n=1701, min 0.062, max 8.585 ms
Throughput before policy	9.44 Gbps	`iperf3` baseline
Throughput after policy	0 (blocked)	No flows observed
Connectivity after policy	100% loss	`ping` failure
Drop rules present	Yes (s1, s4)	OVS dumps confirm entries

Finally, in the *fully decentralized* mode, each domain operates its own complete service stack, writes state into ETCD under its namespace, and coordinates directly with peer FlowBlockers through REST interactions. These modes are selectable via environment variables and script flags, enabling flexible testing and benchmarking of orchestration strategies.

VI. Evaluation and Results

The eMSN framework was evaluated in a two-domain testbed to validate functional correctness and quantify control-plane responsiveness and policy enforcement. The deployment included Ryu (OpenFlow 1.0), the SimpleSwitch REST microservice, and FlowBlocker as the policy engine. Each domain had two Open vSwitch (OVS) datapaths with hosts arranged two per switch (eight total). A three-node ETCD cluster provided shared state. Traffic was generated using Mininet with `ping` and `iperf3`. Instrumentation was additive and non-intrusive, including structured logs, OpenFlow packet captures, domain/flow table snapshots, and OVS dumps.

A. Metrics and Methodology

To assess the behavior of the control plane and the enforcement mechanism, we considered three key measurements.

First, the *Packet-In to Flow-Mod delay* was evaluated. Whenever an OpenFlow switch generates a `packet_in` event, the Ryu emitter captures it and forwards it to the appropriate microservice. The processing microservice (such as SimpleSwitch) then decides on the forwarding action and returns a corresponding `flow_mod` instruction to the Ryu REST client. By timestamping both the arrival of the `packet_in` and the dispatch of the matching `flow_mod`, we measure the responsiveness of the control path. In cases where OpenFlow 1.0 packets indicate `OFP_NO_BUFFER`, we fall back to correlating flows based on source–destination MAC address pairs, ensuring that each `packet_in` is matched to the right `flow_mod`.

Second, the *Policy Request to Processing delay* was examined. This metric captures how quickly a high-level policy submitted by a client is handled inside the system. Each policy `POST` request to FlowBlocker is timestamped, and the interval is measured until FlowBlocker logs its internal decision point (the `POLICY_APPLY` event) and subsequently issues the first successful `flow_mod` to the controller. This reveals the overhead from policy signaling down to actual rule installation.

Third, the *Policy to Enforcement outcome* was validated. We compared network behavior before and after a policy was introduced by observing connectivity and throughput using `ping` and `iperf3`. We also examined the Open vSwitch flow tables directly to confirm the presence of new drop rules. The absence of connectivity, the collapse of throughput, and the explicit drop entries in OVS together verify that policies were enforced correctly.

B. Results

The controller path is responsive: sub-millisecond median Packet-In → Flow-Mod and tens of milliseconds from policy request to enforcement. Enforcement is effective: throughput collapsed to zero and drop entries were confirmed in OVS.

These delays are significant compared to monolithic controllers, where internal coupling and centralized loops often result in increased reaction times. By isolating functions as microservices, eMSN maintains sub-millisecond responsiveness in the control loop while enabling distributed enforcement.

C. Qualitative Comparison with Related Work

To highlight novelty, Table II contrasts eMSN with prior frameworks. Unlike µ-ONOS, MSN, Kandoo, µ-TSN, and MDSO, eMSN uniquely combines decomposition with decentralized multi-domain enforcement using ETCD strictly for state sharing.

VII. Conclusion

This paper presented eMSN, a microservice-based SDN framework for modular and decentralized control across multiple domains. By decomposing controller functions into containerized microservices and coordinating them via ETCD, eMSN enables scalable and domain-aware orchestration without relying on centralized storage of flow rules.

The framework is built around three core components: Ryu Core (emitter and REST client), SimpleSwitch, and FlowBlocker. FlowBlocker enforces domain-specific policies

TABLE II: Qualitative comparison with related frameworks

Feature	MSN	μ-ONOS	μ-TSN	Kandoo	MDSO	**eMSN**
Architecture						
Decomposition	Yes	Yes	Yes	No	No	Yes
Coordination	REST	Central	TSN-spec.	Root	Orchestr.	ETCD+REST
State sharing	No	Internal	Sched.	No	NBI	ETCD (topo/hosts)
Multi-domain & Policy						
Multi-domain support	No	Limited	No	Hier.	Federated	Yes
Policy enforcement	Basic	Central	Determ.	Root	Global	FlowBlocker (local/peer)

through local decision logic and peer-to-peer signaling, ensuring autonomy while sharing only host and topology state through ETCD.

eMSN supports centralized, partially decentralized, and fully decentralized deployments. Functional validation across these modes confirmed low-latency responsiveness, effective policy enforcement, and minimal coupling between domains.

Future work will extend eMSN with large-scale performance evaluation, service-aware and traffic classification policies, and trust-based coordination mechanisms, including AI-assisted orchestration.

ACKNOWLEDGMENT

This work was supported by the European Union - NextGenerationEU under the National Recovery and Resilience Plan (PNRR), Mission 4, Component 2, Investment 1.3 - CUP J33C22002880001 - partnership on "Telecommunications of the Future" (PE00000001 - "RESTART") and Investment 1.1 - PRIN 2022YA59ZJ "ZeTON" - CUP J53D23000930006.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[3] Y. Li and M. Chen, "Software-defined network function virtualization: A survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[4] G. Sticca, M. Ibrahimi, F. Musumeci, N. Di Cicco, A. Castoldi, R. Pastorelli, and M. Tornatore, "Selective hybrid edfa/raman amplifier placement to mitigate lightpath degradation in (c+ l) networks," *Journal of Optical Communications and Networking*, vol. 15, no. 8, pp. C232–C241, 2023.

[5] S. Sharma, N. Saxena, N. Singh, and D. S. Saxena, "A survey of microservice architectures in sdn/nfv environments," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1196–1231, 2019.

[6] H. T. Tran, J. Yoo, and S. Noh, "Microservice-based architecture for the onos sdn controller," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 351–359.

[7] A. Sgambelluri, F. Paolucci, and F. Cugini, "Sdn orchestration for dynamic service chaining with nfv in multi-domain scenarios," in *2017 European Conference on Networks and Communications (EuCNC)*. IEEE, 2017, pp. 1–5.

[8] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 19–24.

[9] D. Barattini, S. Salsano, L. Veltri, and M. Gaggero, "Msn: A microservices-based sdn controller architecture," in *2022 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2022, pp. 500–506.

[10] G. Belocchi, M. Ruffini, and D. Tuncer, "µ-tsn-cp: A microservices-based control plane for time-sensitive networks," in *2021 IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, 2021, pp. 1–6.

[11] X. Guo *et al.*, "Toward intent-driven multi-domain orchestration in sdn," *IEEE Communications Magazine*, vol. 60, no. 3, pp. 72–78, 2022.

[12] Y. Xiang *et al.*, "Intent-based coordination for multi-domain sdn orchestration," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2023, pp. 1–6.